```
// BEE 425 AA, Winter 2021
// Austin Gilbert, Adrian, & Carol Kao, 03/8/21
// Modified from Valvano et al, UTexas & Joseph Decuir, UWashington
// BEE425L21 Lab 4, Version 1: Wave Generator
// main.c
// 1.0 DESCRIPTION:
// ------------------------------------------------------------------------------
// Periodic Waveform Generator with four wave functions (square, ramp, sine,
// triangle), a 10 Hz to 10 kHz variable frequency range, and 32 mV to 8 V
// peak-to-peak amplitude range.
// User selects waveform type via 4x4 Keypad, then presses star. Program changes
// from Keypad-scanning mode to Waveform Display mode, displaying one of the four
// selected waveform types. SW1 and SW2 will then be active. Pressing SW1 will
// allow the user to alter the potentiometer value to change the amplitude of the
// wave. Pressing SW2 allows the user to alter the potentiometer value to change
// the frequency.
// This program also contains a digital filter that averages the current and last
// potentiometer value for a smoother output. The low pass filter will use adaptive
// controls to smooth the output depending on the frequency and upon each waveform.
// 2.0 PRE-PROCESSOR DIRECTIVES SECTION
// ------------------------------------------------------------------------------
// Constant declarations to access port registers using symbolic names
// modified to include TM4C123GH6PM.h & system_TM4C123.h definitions
#include "TM4C123GH6PM.h"       // Keil seems to ignore them
#include "system_TM4C123.h"     // ditto
#include "TExaS.h"

// master port clock
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))

// GPIO Port F
#define GPIO_PORTF_DATA_R       (*((volatile unsigned long *)0x400253FC))
#define GPIO_PORTF_DIR_R        (*((volatile unsigned long *)0x40025400))
#define GPIO_PORTF_AFSEL_R      (*((volatile unsigned long *)0x40025420))
#define GPIO_PORTF_PUR_R        (*((volatile unsigned long *)0x40025510))
#define GPIO_PORTF_DEN_R        (*((volatile unsigned long *)0x4002551C))
#define GPIO_PORTF_LOCK_R       (*((volatile unsigned long *)0x40025520))
#define GPIO_PORTF_CR_R         (*((volatile unsigned long *)0x40025524))
#define GPIO_PORTF_AMSEL_R      (*((volatile unsigned long *)0x40025528))
#define GPIO_PORTF_PCTL_R       (*((volatile unsigned long *)0x4002552C))

// NVIC = Systick
#define NVIC_ST_RELOAD_R        (*((volatile unsigned long *)0xE000E014))
#define NVIC_ST_CTRL_R          (*((volatile unsigned long *)0xE000E010))

// GPIO Port E
#define GPIO_PORTE_DATA_R       (*((volatile unsigned long *)0x400243FC))
#define GPIO_PORTE_DIR_R        (*((volatile unsigned long *)0x40024400))
#define GPIO_PORTE_AFSEL_R      (*((volatile unsigned long *)0x40024420))
#define GPIO_PORTE_PUR_R        (*((volatile unsigned long *)0x40024510))
#define GPIO_PORTE_DEN_R        (*((volatile unsigned long *)0x4002451C))
#define GPIO_PORTE_CR_R         (*((volatile unsigned long *)0x40024524))
#define GPIO_PORTE_AMSEL_R      (*((volatile unsigned long *)0x40024528))
#define GPIO_PORTE_PCTL_R       (*((volatile unsigned long *)0x4002452C))

// ADC AIN0 (PE3)
#define SYSCTL_RCGCADC_R        (*((volatile unsigned long *)0x400FE638))
#define ADC0_ACTSS_R            (*((volatile unsigned long *)0x40038000))
#define ADC0_EMUX_R             (*((volatile unsigned long *)0x40038014))
#define ADC0_SSMUX3_R           (*((volatile unsigned long *)0x400380A0))
#define ADC0_SSCTL3_R           (*((volatile unsigned long *)0x400380A4))
#define ADC0_PSSI_R             (*((volatile unsigned long *)0x40038028))
#define ADC0_SSFIFO3_R          (*((volatile unsigned long *)0x400380A8))
#define ADC0_IM_R               (*((volatile unsigned long *)0x40038008))
#define ADC0_ISC_R              (*((volatile unsigned long *)0x4003800C))
#define ADC0_RIS_R              (*((volatile unsigned long *)0x40038004))
#define ADC0_SSPRI_R            (*((volatile unsigned long *)0x40038020))

// GPIO Port D
#define GPIO_PORTD_DATA_R       (*((volatile unsigned long *)0x400073FC))
#define GPIO_PORTD_DIR_R        (*((volatile unsigned long *)0x40007400))
#define GPIO_PORTD_AFSEL_R      (*((volatile unsigned long *)0x40007420))
```

```
73  #define GPIO_PORTD_PUR_R        (*((volatile unsigned long *)0x40007510))
74  #define GPIO_PORTD_DEN_R        (*((volatile unsigned long *)0x4000751C))
75  #define GPIO_PORTD_CR_R         (*((volatile unsigned long *)0x40007524))
76  #define GPIO_PORTD_AMSEL_R      (*((volatile unsigned long *)0x40007528))
77
78  // GPIO Port C
79  #define GPIO_PORTC_DATA_R       (*((volatile unsigned long *)0x400063FC))
80  #define GPIO_PORTC_DIR_R        (*((volatile unsigned long *)0x40006400))
81  #define GPIO_PORTC_AFSEL_R      (*((volatile unsigned long *)0x40006420))
82  #define GPIO_PORTC_PUR_R        (*((volatile unsigned long *)0x40006510))
83  #define GPIO_PORTC_DEN_R        (*((volatile unsigned long *)0x4000651C))
84  #define GPIO_PORTC_CR_R         (*((volatile unsigned long *)0x40006524))
85  #define GPIO_PORTC_AMSEL_R      (*((volatile unsigned long *)0x40006528))
86
87  // GPIO Port B
88  #define GPIO_PORTB_DATA_R       (*((volatile unsigned long *)0x400053FC))
89  #define GPIO_PORTB_DIR_R        (*((volatile unsigned long *)0x40005400))
90  #define GPIO_PORTB_AFSEL_R      (*((volatile unsigned long *)0x40005420))
91  #define GPIO_PORTB_PUR_R        (*((volatile unsigned long *)0x40005510))
92  #define GPIO_PORTB_DEN_R        (*((volatile unsigned long *)0x4000551C))
93  #define GPIO_PORTB_CR_R         (*((volatile unsigned long *)0x40005524))
94  #define GPIO_PORTB_AMSEL_R      (*((volatile unsigned long *)0x40005528))
95
96
97  // 3.0 DECLARATIONS SECTION
98  // ---------------------------------------------------------------------------
99  //   Global Variables
100 int Mode;                                   // 0 = wave, 1 = scan
101 int SW1;                                    // first switch
102 int SW2;                                    // second switch
103 int WaveM;                                  // waveform mode: 1 sine, 2 ramp,
104                                             // 3 triangle, 4 square
105 int AdjustM;                                // adjustment mode: 0 amplitude, 1 frequency
106 int DAC;                                    // DAC output
107 int SWI;                                    // Sine Wave index: 12 entries
108 int RWI;                                    // Ramp Wave index: 16 entries
109 int TWI;                                    // Triangle Wave index: 12 entries
110 int SQWI;                                   // Sqare wave index: 2 entries
111
112 // 12 sample signwave table
113 int SinT[] = {0x80,0xC0,0xEE,0xFF,0xEE,0xC0,0x80,0x40,0x12,0x00,0x12,0x40};
114 // 16 sample rampwave table
115 int RampT[] = {0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0xA0,
116                0xB0, 0xC0, 0xD0, 0xE0, 0xF0};
117 //int RampT[] = {0x00, 0xF0};
118 // 12 sample trianglewave table
119 //int TriT[] = {0x80,0xAB,0xD5,0xFF,0xD5,0xAB,0x80,0x55,0x2B,0x00,0x2B,0x55};
120 int TriT[] = {0x80,0xFF,0x80,0x00};
121 int SquareT[] = {0x00, 0xFF};               // 2 sample squarewave table
122 int KeyColCtr;                              // keyboard column counter
123 int KeyCol;                                 // Keyboard column output
124 int KeyColIndex;                            // Keyboard column index
125 int KeyColumn;                              // Keyboard column drive
126 int KeyRow;                                 // Keyboard row input
127 int KeyRowIndex;                            // Keyboard row
128 int KeyCode;                                // 4x4 key location
129 int KeyHex;                                 // encoded hex value
130 int LastKeyHex;                             // last encoded hex value
131 int KeyDetect;                              // emperical key detect
132 int KeyString;                              // the 8-bit string of each 4-bit digit
133 int LPFM;                                   // Low Pass Filter Mode: 0-3 on; 4-7 off
134 int SM;                                     // Small Mode: 1 on; 0 off
135 volatile int AIN0;                          // input from 12-bit ADC on PE3
136 double pot;                                 // double casted input from 12-bit ADC on PE3
137 double potRatio;                            // potentiometer value as a ratio/percent
138 double ampRatio;                            // last potentiometer ratio value for amplitude
139 double freqRatio;                           // last potentiometer ratio value for frequency
140 double outDouble;                           // double value for waveforms
141 double maxDouble;                           // double value for maximum timing (used in 5.8)
142 int min;                                    // long value for minimum time
143 int max;                                    // long value for maximum time
144 unsigned long ColorCount;                   // color counter
```

```
145   volatile unsigned long time;                // long value for NVIC value
146
147   //   Function Prototypes
148   void Delay(void);
149   void NVIC_Init(int);                        // added NVIC function, variable value
150   void PortF_Init(void);                      // set up Port F for Switches and LEDs
151   void KBD_Init(void);                        // set up Port C & D for 4x4 keyboard
152   void DAC_Init(void);                        // set up Port B for DAC
153   void ADC_Init(void);                        // add ADC0 function on PE3
154   int Output(int, double);                    // turns wave table output into
155                                               // amplitude-adjustable output
156   int Timing(int, int, double);            // takes min and max and outputs
157                                               // pot-adjusted value
158
159   // 4.0 MAIN CODE BLOCK
160   // ---------------------------------------------------------------------------
161   // MAIN: Mandatory for a C Program to be executable
162   int main(void){
163     NVIC_Init(15999);                         // Call Systick initialization to 1 msec
164     PortF_Init();                             // Call initialization of SW and LEDs
165     ADC_Init();                               // call ADC initialization
166     DAC_Init();                               // Call initialization of DAC port B
167     KBD_Init();                               // Call initialization of 4x4 keyboard
168
169     while(1){
170
171       // KEYPAD SCANNING MODE
172       ////////////////////////////////////////////////////////////////////////////
173       if (Mode == 0) {
174         GPIO_PORTF_DATA_R = 0x02;             // Turn LED red (Port F)
175         GPIO_PORTB_DIR_R = 0x3F;              // turn off PB7-6; keep PB5-0 on
176         time = 15999;
177
178         // Keypad Scan - PC to PD - result to PB and PF
179         Delay();                              // wait specfied time (1ms)
180         if (KeyDetect == 0) KeyColCtr +=1;    // count by 1
181         KeyColIndex = (KeyColCtr) & 0x3;      // index should be 0, 1, 2 or 3
182         if (KeyColIndex==0) KeyColumn = 0xE0; // ground PC4: A, B, C, D
183         if (KeyColIndex==1) KeyColumn = 0xD0; // ground PC5: 3, 6, 9, #\F
184         if (KeyColIndex==2) KeyColumn = 0xB0; // ground PC6: 2, 5, 8, 0
185         if (KeyColIndex==3) KeyColumn = 0x70; // ground PC7: 1, 4, 7, *\E
186         GPIO_PORTC_DATA_R = KeyColumn;        // drive column ports
187
188         // read row ports
189         Delay();                              // wait for ports to settle
190         KeyRow = (GPIO_PORTD_DATA_R & 0x0F);  // capture key row
191
192         // detect and process keys
193         if (KeyRow==0x0F) {                   // no key found
194           GPIO_PORTF_DATA_R = 0x02;           // keep indicator RED
195           KeyRowIndex = 0x4;                  // indicating not found
196           KeyDetect = 0;
197           LastKeyHex = KeyHex;                // Store last key hex
198
199         } else {
200           KeyDetect = 1;                      // key found
201           GPIO_PORTF_DATA_R = 0x04;           // set indicator BLUE
202           if (KeyRow==0xE) KeyRowIndex = 0x0; // row 0: D, #\F, 0, *\E
203           if (KeyRow==0xD) KeyRowIndex = 0x1; // row 1: C, 9, 8, 7
204           if (KeyRow==0xB) KeyRowIndex = 0x2; // row 2: B, 6, 5, 4
205           if (KeyRow==0x7) KeyRowIndex = 0x3; // row 3: A, 3, 2, 1
206           KeyCode = (KeyColIndex << 2)+ KeyRowIndex; // assemble key code
207
208           // GPIO_PORTB_DATA_R = KeyCode;   output key code
209           // insert hex encoder: 16 key codes in, 16 hex values out
210           if(KeyCode==0)  KeyHex = 0xD;
211           if(KeyCode==1)  KeyHex = 0xC;
212           if(KeyCode==2)  KeyHex = 0xB;
213           if(KeyCode==3)  KeyHex = 0xA;
214           if(KeyCode==4)  KeyHex = 0xF;       // map # to 15
215           if(KeyCode==5)  KeyHex = 0x9;
216           if(KeyCode==6)  KeyHex = 0x6;
```

```
217           if(KeyCode==7)  KeyHex = 0x3;
218           if(KeyCode==8)  KeyHex = 0x0;
219           if(KeyCode==9)  KeyHex = 0x8;
220           if(KeyCode==10) KeyHex = 0x5;
221           if(KeyCode==11) KeyHex = 0x2;
222           if(KeyCode==12) KeyHex = 0xE;       // map * to 14
223           if(KeyCode==13) KeyHex = 0x7;
224           if(KeyCode==14) KeyHex = 0x4;
225           if(KeyCode==15) KeyHex = 0x1;
226
227           // Create KeyString
228           KeyString = KeyHex | ((LastKeyHex << 4) & 0xF0);
229           Delay();
230           GPIO_PORTB_DATA_R = KeyString & 0xFF; // Output to Port B
231
232           // Pressing Last Key "Star"
233           if ((KeyString & 0xF) == 0xE) {
234             // Pressing First Key 1
235             if (((KeyString >> 4) & 0xF) == 0x1) {
236               WaveM = 1;                      // Activate Sine Wave Mode
237               Mode = 1;                       // Branch to else loop
238
239             // Pressing First Key 2
240             } else if (((KeyString >> 4) & 0xF) == 0x2) {
241               WaveM = 2;                      // Activate Sine Wave Mode
242               Mode = 1;                       // Branch to else loop
243
244             // Pressing First Key 3
245             } else if (((KeyString >> 4) & 0xF) == 0x3) {
246               WaveM = 3;
247               Mode = 1;                       // Branch to else loop
248
249             // Pressing First Key 4
250             } else if (((KeyString >> 4) & 0xF) == 0x4) {
251               WaveM = 4;
252               Mode = 1;                       // Branch to else loop
253
254             // Pressing any other Key
255             } else {
256               int i;
257               NVIC_Init(7999999);             // change timing for flashing
258               // Flash RED LED
259               for (i = 0; i <= 2; ++i) {      // for loop to flash twice
260                 GPIO_PORTF_DATA_R = 0x00;
261                 Delay();
262                 GPIO_PORTF_DATA_R = 0x02;
263                 Delay();
264               }
265               NVIC_Init(15999);               // change timing back for key-scanning
266               Mode = 0;                       // Restart Mode 0 loop
267             }
268           }
269         }
270
271       // WAVEFORM DISPLAY MODE
272       ////////////////////////////////////////////////////////////////////////////
273       } else if (Mode == 1) {
274         // Initialize Ports & Timing
275         NVIC_Init(time);
276         GPIO_PORTB_DATA_R = DAC;              // DAC = Port B Output
277         GPIO_PORTE_DATA_R = LPFM;             // LPFM = Port E Output
278         SW1 = GPIO_PORTF_DATA_R & 0x10;       // sample port PF4
279         SW2 = GPIO_PORTF_DATA_R & 0x01;       // sample port PF0
280         GPIO_PORTB_DIR_R = 0xFF;              // restore PB7-PB0 output
281
282         // Initialize Small Mode
283         if (time < 799) {
284           SM = 1;
285         }
286
287         // Initialize Switches
288         if (SW1 == 0) AdjustM = 0;
```

```
289         if (SW2 == 0) AdjustM = 1;
290
291         // Variable LPFM
292         if (WaveM == 1 || WaveM == 3) { // Sine or Triangle wave
293           if (time <= 133332 && time >= 3366) {
294             LPFM = 1;                           // 10 Hz - 396 Hz
295           } else if (time < 3366 && time >= 336) {
296             LPFM = 2;                           // 3.96 kHz - 396 Hz
297           } else if (time < 336 && time >= 132) {
298             LPFM = 3;                           // 10 kHz - 3.96 kHz
299           }
300         } else if (WaveM == 2) { // Ramp Wave
301           if (time <= 99999 && time >= 2524) {
302             LPFM = 1;                           // 10 Hz - 396 Hz
303           } else if (time < 2524 && time >= 252) {
304             LPFM = 2;                           // 3.96 kHz - 396 Hz
305           } else if (time < 252 && time >= 99) {
306             LPFM = 3;                           // 10 kHz - 3.96 kHz
307           }
308         } else { // WaveM == 4
309           LPFM = 7;                             // LPF off
310         }
311
312         // Capture Current ADC Input
313         ADC0_PSSI_R |= 8;                       // 7) start a conversion at sequence 3
314         while((ADC0_RIS_R & 8) == 0);           // 8) wait for conversion to complete
315         AIN0 = ADC0_SSFIFO3_R;                  // 9) capture the results
316         pot = ((AIN0 >> 4) & 0xFF) * 1.0;       // get 8-bit potentiometer value
317         potRatio = pot / 255.0;                 // ratio of potentiometer turn
318         ADC0_ISC_R = 8;                         // 10) clear completion flag
319         Delay();                                // wait specified time (1ms)
320         // Step All Wave-types
321         SWI += 1;                               // step sine wave index
322         if(SWI == 12) SWI = 0;                  // counts up 12 times
323         RWI += 1;                               // step ramp wave index
324         if(RWI == 16) RWI = 0;                  // counts up 16 times
325         TWI+=1;                                 // step triangle wave index
326         if(TWI == 4)  TWI = 0;                  // counts up 12 times
327         SQWI += 1;                              // step square wave index
328         if(SQWI == 2) SQWI = 0;                 // counts up 2 times
329
330         // AMPLITUDE ADJUSTMENT MODE
331         if (AdjustM == 0) {
332           GPIO_PORTF_DATA_R = 0x08;             // Turn LED GREEN
333           ampRatio = potRatio;                  // save amplitude ratio
334
335           // Sine Wave
336           if (WaveM == 1) {
337             if (freqRatio != 0) {               // keep freq value
338               time = Timing(133332, 132, freqRatio);
339             }
340             DAC = Output(SinT[SWI], potRatio); // output sine
341           // Ramp Wave
342           } else if (WaveM == 2) {
343             if (freqRatio != 0) {               // keep freq value
344               time = Timing(99999, 99, freqRatio);
345             }
346             DAC = Output(RampT[RWI], potRatio); // output ramp
347           // Triangle Wave
348           } else if (WaveM == 3) {
349             if (freqRatio != 0) {               // keep freq value
350               time = Timing(399999, 399, freqRatio);
351             }
352             DAC = Output(TriT[TWI], potRatio); // output tri
353           // Square Wave
354           } else if (WaveM == 4) {
355             if (freqRatio != 0) {               // keep freq value
356               time = Timing(799999, 799, freqRatio);
357             }
358             DAC = Output(SquareT[SQWI], potRatio); // output square
359           }
360
```

```
361         // FREQUENCY ADJUSTMENT MODE
362         } else { // AdjustM = 1
363           GPIO_PORTF_DATA_R = 0x0C;           // Turn LED SKY BLUE
364           freqRatio = potRatio;               // save frequency ratio
365 
366           // Sine Wave
367           if (WaveM == 1) {
368             time = Timing(133332, 132, potRatio); // change freq via pot
369             if (SM == 1 && time < 5332) {
370               time = Timing(799999, 799, potRatio);
371               DAC = Output(SquareT[SQWI], ampRatio);
372             } else {
373               SM = 0;
374               DAC = Output(SinT[SWI], ampRatio); // output sine
375             }
376           // Ramp Wave
377           } else if (WaveM == 2) {
378             time = Timing(99999, 99, potRatio); // change freq via pot
379             if (SM == 1 && time < 5332) {
380               time = Timing(799999, 799, potRatio);
381               DAC = Output(SquareT[SQWI], ampRatio);
382             } else {
383               SM = 0;
384               DAC = Output(RampT[SWI], ampRatio); // output ramp
385             }
386           // Triangle Wave
387           } else if (WaveM == 3) {
388             time = Timing(133332, 132, potRatio); // change freq via pot
389             if (SM == 1 && time < 5332) {
390               time = Timing(799999, 799, potRatio);
391               DAC = Output(SquareT[SQWI], ampRatio);
392             } else {
393               SM = 0;
394               DAC = Output(TriT[SWI], ampRatio); // output tri
395             }
396           // Square Wave
397           } else if (WaveM == 4) {
398             time = Timing(799999, 799, potRatio); // change freq via pot
399             DAC = Output(SquareT[SQWI], ampRatio); // output square
400           }
401         }
402       // Avoiding possible bugs
403       } else {
404         Mode = 0;
405       }
406     }
407   }
408 
409 
410   // 5.0 SUBROUTINE FUNCTIONS
411   // -------------------------------------------------------------------------------
412   // 5.1 Initialize Systick timer
413   //    int time -- the int value to use for NVIC time
414   void NVIC_Init(int time){
415     volatile unsigned long Systick;
416     NVIC_ST_RELOAD_R = time;          // configure Systick with value "time"
417     NVIC_ST_CTRL_R = 5;               // configure Systick for auto reload
418   }
419   // 5.2 initialize port F pins for input and output
420   // PF4 and PF0 are inputs SW1 and SW2 respectively
421   // PF3,PF2,PF1 are outputs to the LED
422   void PortF_Init(void){ volatile unsigned long delay;
423     SYSCTL_RCGC2_R |= 0x00000020;     // 1) PF clock
424     delay = SYSCTL_RCGC2_R;           // delay
425     GPIO_PORTF_LOCK_R = 0x4C4F434B;   // 2) unlock PortF
426     GPIO_PORTF_CR_R = 0x1F;           // allow changes to PF4-0
427     GPIO_PORTF_AMSEL_R = 0x00;        // 3) disable analog function
428     GPIO_PORTF_PCTL_R = 0x00000000;   // 4) GPIO clear bit PCTL
429     GPIO_PORTF_DIR_R = 0x0E;          // 5) PF4,PF0 input, PF3,PF2,PF1 output
430     GPIO_PORTF_AFSEL_R = 0x00;        // 6) no alternate function
431     GPIO_PORTF_PUR_R = 0x11;          // 7) enable pullup resistors on PF4,PF0
432     GPIO_PORTF_DEN_R = 0x1F;          // 8) enable digital pins PF4-PF0
```

```
433   }
434   // 5.3 initialize keyboard ports: PC7-4 columns; PD3-0 rows; PD4-0 code results
435   void KBD_Init(void) {;
436     SYSCTL_RCGC2_R |= 0x0C;           // 1) PC & PD clocks
437     GPIO_PORTC_AMSEL_R = 0x00;        // 2) disable analog function
438     GPIO_PORTC_AFSEL_R = 0x00;        // 3) no alternate function
439     GPIO_PORTC_DEN_R |= 0xF0;         // 4) enable digital pins PC7-PC4
440     GPIO_PORTC_DIR_R |= 0xF0;         // 5) PC7-PC4 output - turn off outputs
441     GPIO_PORTD_AMSEL_R = 0x00;        // 6) disable analog function
442     GPIO_PORTC_AFSEL_R = 0x00;        // 7) no alternate function
443     GPIO_PORTD_DEN_R |= 0xCF;         // 8) enable digital pins PD3-PD0
444     GPIO_PORTD_DIR_R &= 0x00;         // 5) PD3-PD0 input
445     GPIO_PORTD_PUR_R = 0x0F;          // 7) enable pullup resistors on PD3-PD0
446   }
447   // 5.4 initialize DAC output port: PB7-0
448   void DAC_Init(void)  {;
449     SYSCTL_RCGC2_R |= 0x02;           // 1) PB clock
450     GPIO_PORTB_AMSEL_R = 0x00;        // 2) disable analog function
451     GPIO_PORTB_AFSEL_R = 0x00;        // 3) no alternate function
452     GPIO_PORTB_DEN_R = 0xFF;          // 4) enable digital pins PB7-PB0
453     GPIO_PORTB_DIR_R = 0xFF;          // 5) PB7-PB0 output
454   }
455   // 5.5 initialize ADC input on PE3 - compare to Mazidi Ch7 p187, P7-1
456   void ADC_Init(void){volatile unsigned long AIN0;
457     SYSCTL_RCGC2_R |= 0x10;           // 0) E clock enable
458     SYSCTL_RCGCADC_R |= 1;            // 1) enable clock for ADC0
459     GPIO_PORTE_AFSEL_R |= 8;          // enable alternate function on PE3
460     GPIO_PORTE_DEN_R = 7;             // disable digital pin PE3, enable PE2-PE0
461     GPIO_PORTE_AMSEL_R |= 8;          // enable analog function PE3
462     GPIO_PORTE_DIR_R = 7;             // PE3 input, PE2,PE1,PE0 output
463     ADC0_ACTSS_R &= ~8;               // 2) disable SS3 during configuration
464   //  SYSCTL_RCGC2_R |= 0x00010000;      //  activate ADC0                    ?
465   //  GPIO_PORTE_PCTL_R = 0x0000;       // GPIO clear bit PCTL
466   //  SYSCTL_RCGC2_R &= ~0x00000300;    //  configure for 125K sample rate    ?
467   //  ADC0_SSPRI_R = 0x0123;            // sequencer 3 is highest priority
468     ADC0_EMUX_R &= ~0xF000;           // 3) software trigger conversion
469     ADC0_SSMUX3_R = 0;                // 4) select input from channel 0
470     ADC0_SSCTL3_R |= 6;               // 5) sample and set time at 1st sample
471     ADC0_IM_R |= (1<<3);                // 14) enable imterrupt mask for SS3
472     ADC0_ACTSS_R |= 8;                // 6) enable ADC0 sequencer 0
473   }
474   // 5.6 Delay function, using Systick,
475   void Delay(void){
476     while ((NVIC_ST_CTRL_R & 0x10000) == 0);
477   }
478   // 5.7 Wave output function
479   //    int waveIn      -- the hex input wave value
480   //    double potRatio -- the potentiometer ratio value to use as a percentage
481   //      returns       -- the pot-adjusted output int
482   int Output(int waveIn, double potRatio) {
483     outDouble = (waveIn - 0x80) * 1.0;
484     outDouble *= potRatio;
485     return (int)outDouble + 0x80;
486   }
487   // 5.8 Frequency Timing function
488   //    long max        -- max time value
489   //    long min        -- min time value
490   //    double potRatio -- the potentiometer ratio value to use as a percentage
491   //      returns       -- the pot-adjusted output long
492   int Timing(int max, int min, double potRatio) {
493     max -= min;
494     maxDouble = (double)max;
495     maxDouble *= potRatio;
496     return (int)maxDouble + min;
497   }
498
```